DISEÑO DE FILTROS Y SU ACELERACIÓN PARA EL ESTÁNDAR H.265/HEVC

SORIN DRĂGHICI

GRADO EN INGENIERÍA INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Grado en Ingeniería Informática

2016 - 2017

Directores:

Carlos García Sánchez, Guillermo Botella Juan

Autorización de Difusión

SORIN DRĂGHICI

9-abril-2017

El/la abajo firmante, matriculado/a en el Grado en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Grado: "DISEÑO DE FILTROS Y SU ACCELERACIÓN PARA EL ESTÁNDAR H.265/HEVC", realizado durante el curso académico 2016-2017 bajo la dirección de Carlos García Sánchez, Guillermo Botella Juan en el departamento Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

Entre los principales impactos producidos por el incremento de la calidad de vídeo se encuentran: i) la gran cantidad de espacio físico necesario para el almacenamiento de estos debido a la alta cantidad de píxeles almacenados por fotograma, y ii) el alto ancho de banda necesario por unidad de tiempo y capacidad de procesamiento para la decodificación de estos archivos en internet u offline.

La solución inmediata que se nos presenta es la codificación de vídeo, cuyo principal objetivo es obtener una proporción aceptable entre espacio y calidad. La codificación por sí sola puede no resultar suficiente debido a la formación previa del vídeo a codificar, el cual suele presentar desigualdades y grandes cambios innecesarios de intensidad entre píxeles muy cercanos, lo cual provoca que en la codificación no se obtenga el mejor resultado. Como solución a lo mencionado, se realiza un filtrado previo del vídeo para dar uniformidad a los fotogramas y eliminar las grandes diferencias entre los píxeles para facilitar el trabajo de codificación.

Tanto el filtrado como la codificación requieren de hardware de alto rendimiento para llevarse a cabo, por ello, se hará uso del paradigma de la computación paralela con OpenCL para disminuir favorablemente los tiempos de ejecución.

Se obtendrán distintas métricas de calidad entre los vídeos para decidir cuál es el mejor resultado obtenido.

Como conclusión, los pasos seguidos en el trabajo han sido la implementación de varios filtros sobre vídeos YUV tanto en software como en hardware con OpenCL, uso del estándar H.265/HEVC como caja negra para codificar los vídeos obtenidos en la fase de filtrado, y finalmente, implementación de métricas de medición calidad de vídeo, para poder comparar los distintos resultados obtenidos, todo ello en tiempo real.

Palabras clave

H265, HEVC, filtrado, codificación, métrica de calidad, OpenCL, YUV.

Resumen en inglés

Some of the main issues provoked by the increment in video quality are: i) the extent of physical space needed to save these videos due to the great amount of pixel concentration on each frame, and ii) the high bandwidth and processing capacity required to decode these files in real time online or offline.

The immediate solution presented to us is video coding, whose main objective is to obtain an acceptable proportion between space and quality. Coding just by itself may not always be sufficient or effective due to the formation of the video, which will most likely present inequalities, unexpected and unnecessary pixel intensity changes on neighboring pixels, and in consequence the coding will not obtain the best result. The solution here is filtering the video before coding to give it uniformity and remove the unnecessary neighboring pixel changes mentioned before, which will make the coding job easier.

Both filtering and coding are complex operations, that is why they require high end hardware, so the parallel computing paradigm with OpenCL will be used to lower the execution times.

Different quality metrics will be used to decide which is the best outcome obtained.

Concluding, the main steps followed were, implementing multiple filters for videos on YUV space both on software and hardware using OpenCL, use of the H.265/HEVC standard as a black box to encode the videos obtained in the previous phase, and finally, implementation of video quality metrics to compare the differences between the obtained videos, all of which is done in real time.

Keywords

H265, HEVC, filter, coding, quality metric, OpenCL, YUV.

Índice

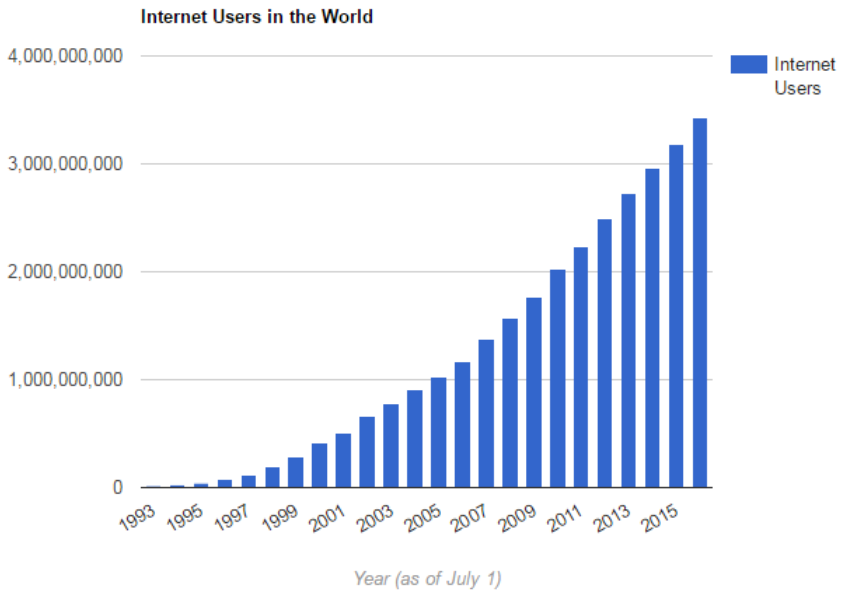
Capítulo 1 – Introducción

Motivación

En la actualidad, debido a los avances tecnológicos producidos a lo largo de la última década y en la que nos encontramos, es de lo más común que las personas estemos en posesión de varios dispositivos capaces de reproducir contenido multimedia.

Con la aparición de los smartphones a mediados-finales de la década anterior, la cantidad de usuarios que hacen uso de éstos no ha dejado de incrementar, llegando a sobrepasar los 2 billones de personas en posesión de un dispositivo móvil circa 2016, siendo esto una cantidad mayor que la de dispositivos de escritorio.

Este cambio que acabamos de mencionar no habría sido posible sin la facilidad con la que se puede conseguir acceso a internet. A principios de los 2000 apenas un 5% de la población mundial tenían acceso a internet, pero esto cambió radicalmente al mismo tiempo que la aparición del smartphone, cuando se sobrepasó el billón de personas con conexión a la red, estando hoy en día en los 3.7 billones y creciendo (esto es aproximadamente la mitad de la población mundial).



Evolución de la cantidad de usuarios de internet.[1]

[1] https://producaoindustrialblog.wordpress.com/2016/12/25/internet-users/

Entre la amplia cantidad de funciones aportadas por un dispositivo móvil, una de las principales es la visualización de vídeo, la cual resulta ser una de las más costosas en términos de ancho de banda debido a todo el flujo de bits que esto supone.

Metodología

La idea principal detrás del presente trabajo es el estudio del funcionamiento de los codificadores de vídeo y los resultados que se obtienen en función de distintos filtros aplicados en una fase previa a la codificación, para obtener un aceptable flujo de bits en los dispositivos, manteniendo al mismo tiempo una calidad y espacio ocupado razonables.

Para ello, se implementarán algunos filtros de vídeo en formato YUV, los vídeos serán codificados para realizar la compresión de los mismos, y posteriormente, la implementación de varias métricas de calidad de vídeo para poder medir los resultados obtenidos.

En concreto, uno de los formatos de decodificación de vídeo más popular es el formato "Planar 4:2:0 YUV", (el cual será explicado en el siguiente capítulo), y uno de los estándares de codificación/decodificación más eficientes y actuales es el H.265/HEVC, razones por las cuales se ha decidido hacer todo el estudio y trabajo presente.

Capítulo 2 – Visión por Computador

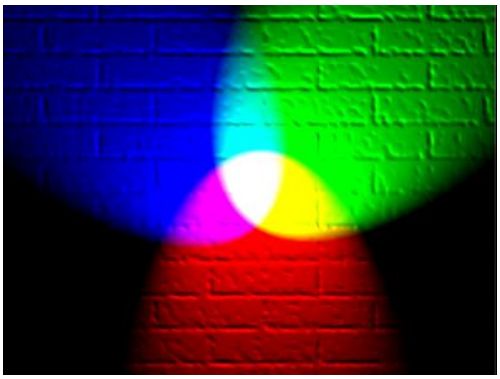
Transformaciones de color

RGB

Originalmente las imágenes digitales, también conocidas como fotogramas cuando forman parte de un vídeo, están compuestas por tres componentes denominados RGB, del inglés Red, Green, Blue, los tres combinados permiten dar color a las imágenes. Los fotogramas están formados por píxeles y cada pixel a su vez puede tomar un valor digital comprendido entre [0, 255], conocido como unidad “unsigned char” o “byte”, formado por 8 bits.

Si consideramos un fotograma de 100 píxeles de ancho y 100 píxeles de alto (ANCHO x ALTO a partir de ahora), en principio podríamos pensar que en total obtendríamos 10.000 píxeles (100x100), lo cual sería cierto si se tratara de una imagen a escala de grises puesto que los píxeles únicamente pueden tomar valores a escala de grises, siendo 0 completamente negro, y 255 completamente blanco (es decir, el valor que el pixel toma es su intensidad entre negro y blanco).

Consecuentemente cada uno de los canales que forman los componentes RGB anteriormente mencionados, por separado forman imágenes a escala de grises, y el color se obtiene al mezclar los valores de la intensidad de cada píxel de cada canal (e.g. la combinación de R = 0, G = 0, B = 255, daría como resultado un azul muy intenso puesto que los componentes de rojo y verde se anulan y el azul toma su máximo valor). A causa de esto, en una imagen RGB de 100x100 a color no se obtendrían 10.000 píxeles, sino 3 x 10.000, es decir, 10.000 píxeles por cada canal de color, y en total 30.000 píxeles que darían lugar a la imagen en color.



Podemos observar que RGB nos permite obtener cualquier color.[2]

Resumiendo, puesto que los píxeles únicamente pueden tomar valores a escala de grises y no de color, nos vemos obligados a guardar más información de la necesaria (en concreto el triple de información) para obtener la sensación de color digital. Es aquí donde entra en juego el espacio de color YUV.

YUV

YUV, al igual que RGB, es un espacio de color que permite guardar información de color, pero teniendo en cuenta la percepción humana de la luminosidad (luma) frente a la crominancia (chroma) la cual es reducida.

El ojo humano es muy sensible a los cambios de luminosidad, Y, o luma, la cual representa el brillo de la imagen a escala de grises, y se puede obtener a partir de RGB mediante Y' = 0.299R + 0.587G + 0.114B.

Los componentes UV representan la crominancia, es decir el color. El ojo humano es menos sensible a los cambios es estos componentes, por ello, la información guardad para estos componentes es menor que la de la luma. Al igual que ocurre en RGB, cada una de estas componentes están a escala de grises por sí solas, pero su combinación da lugar al color. Se pueden obtener de la siguiente forma: U = 0.492(B – Y') esto es la diferencia azul de chroma también conocida como Cb; V = 0.877(R – Y'), esto es la diferencia roja de chroma también conocida como Cr.

A continuación, se puede visualizar un ejemplo de luminancia (Y), crominancia (Cb + Cr), y el resultado final, respectivamente de izquierda a derecha:[3]

[2] http://englishdictionary.education/en/rgb

[3] https://en.wikipedia.org/wiki/Chrominance

Llegado a este punto se podría pensar que en YUV ocurre lo mismo que en RGB, se sigue guardando el triple de información, para cada componente YCbCr, pero esto no es del todo cierto porque YUV introduce un nuevo término denominado "subsampling" el cual como se ha dicho anteriormente permite guardar menos cantidad de píxeles de crominancia mientras que la cantidad de luminancia se mantiene. Sí es cierto que un caso de subsampling (en concreto, el subsampling 4:4:4 como se verá posteriormente) guarda la misma cantidad de información que RGB, en este caso no se obtendrá un beneficio de cantidad de información guardada para la imagen.

Subsampling en YUV

El subsampling, o más en concreto "chroma subsampling" está formado por 3 o 4 unidades numéricas que indican en qué proporción aparecen los tres componentes YCbCr en el espacio YUV.

Es de la forma J:a:b:alpha, donde "J" indica el ancho en píxeles de la región mínima de muestras tomadas (normalmente este ancho suele ser 4 píxeles, además vamos a suponer que el alto siempre es de 2 píxeles), "a" indica la cantidad de muestras de Cb y Cr en la primera fila de la región, "b" indica la cantidad de cambios producidos de Cb y Cr entre la primera y segunda fila de la región de muestras, y finalmente el canal "alpha" sirve para combinar la imagen con otra imagen que se encuentre de fondo para producir el efecto de transparencia, la cual vamos a omitir en este caso. Se tienen varios casos de subsampling, a continuación, vamos a cubrir solo algunos.

4:2:0

Este es el caso más común de chroma subsampling, y es el utilizado para el procesamiento de vídeos CIF en el presente TFG. El subsampling nos indica que, en una muestra de 4 píxeles de ancho, en la primera fila se tiene únicamente 2 muestras de píxeles de crominancia, y en la segunda

fila se tiene 0 cambios con respecto a la primera fila (en otras palabras, la muestra de la segunda fila es la misma que la primera fila). La resolución horizontal es de 1/2, porque se tiene 2 de 4 (2/4 = 1/2) muestras horizontales, y la resolución vertical también es de 1/2, porque se tiene 1 de 2 muestras verticales.

Para entender más en detalle el subsampling, supongamos una imagen de 100x100 de la cual decíamos anteriormente en RGB que tenemos que guardar 3 x 10.000 píxeles. Pues bien, en el espacio YCbCr se guardarán:

- 100x100 = 10.000 píxeles para la luma (a la luma no se le aplica subsampling como se mencionó anteriormente)
- (1/2 x 100) x (1/2 x 100) = 1/4 (100x100) = 2.500 píxeles para Cb. En este caso se ha aplicado 1/2 tanto a la resolución horizontal como a la vertical de la componente Cb.
- (1/2 x 100) x (1/2 x 100) = 1/4 (100x100) = 2.500 píxeles para Cr. De nuevo se ha aplicado 1/2 tanto a la resolución horizontal como a la vertical de la componente Cr.

Finalmente, sumando los valores obtenidos (100x100) + 1/4 (100x100) + 1/4 (100x100) = 3/2 (100x100) podemos llegar a varias conclusiones:

1. El valor final que el subsampling toma en el caso 4:2:0 es de 3/2, si multiplicamos este valor por el WxH de la imagen, se obtiene la cantidad final de píxeles que se guardarán.
2. La cantidad final de píxeles que se guardarán en YUV 4:2:0 es de 3/2 x (100x100) = 15.000, frente a los 30.000 en RGB. Esto es un 100% – (3/2 * 100% / 3) = 50% menos de píxeles guardados por fotograma.
3. Aunque en YUV 4:2:0 se pierdan muchos píxeles de color, la diferencia prácticamente no es apreciable, como se puede ver a continuación:

RGB YUV 4:2:0[4]

4:4:4

Al finalizar el apartado "YUV" se mencionó que en el caso del subsampling 4:4:4 se guarda la misma cantidad de píxeles que en el espacio RGB, vamos a ver por qué ocurre así.

El subsampling nos indica que, en una muestra de 4 píxeles de ancho, en la primera fila se tiene 4 muestras de píxeles de crominancia, y en la segunda fila se tiene 4 cambios con respecto a la primera fila (en otras palabras, la muestra de la segunda fila es totalmente distinta a la primera fila). La resolución horizontal es de 1 (resolución total), porque se tiene 4 de 4 (4/4 = 1) muestras horizontales, y la resolución vertical también es de 1, porque se tiene 2 de 2 muestras verticales.

Realizando los mismos cálculos que en el apartado "4:2:0" con una imagen de 100x100, se llega a una cantidad final de (100x100) + (1x100 x 1x100) + (1x100 + 1x100) = 3 (100x100) = 30.000 píxeles y subsampling = 3. Al igual que en RGB, con este subsampling de 3, no se pierde nada de información.

Otros subsamplings

- 4:2:2 La resolución horizontal es de 1/2 y la vertical es de 1. El valor final que toma el subsampling es 2, porque (WxH) + (1/2xW x 1xH) + (1/2xW + 1xH) = 2 x (WxH). Se guardarán 100 – (2 * 100 / 3) = 33.33% menos de píxeles por fotograma.
- 4:4:0 La resolución horizontal es de 1 y la vertical de 1/2. El valor final que toma el subsampling es 2, porque (WxH) + (1xW x 1/2xH) + (1xW + 1/2xH) = 2 x (WxH). Se guardarán 100 – (2 * 100 / 3) = 33.33% menos de píxeles por fotograma.

[4] https://en.wikipedia.org/wiki/Lenna

- 4:1:1 La resolución horizontal es de 1/4 y la vertical de 1. El valor final que toma el subsampling es 3/2, porque (WxH) + (1/4xW x 1xH) + (1/4xW + 1xH) = 3/2 x (WxH). Se guardarán 100 – (3/2 * 100 / 3) = 50% menos de píxeles por fotograma.

Diseño de filtros específicos

Se conoce como filtrado de imágenes a la modificación de cada uno de los píxeles de una imagen, aunque no siempre pero normalmente haciendo uso del entorno de vecindad de píxeles, con el fin de obtener la misma imagen modificada de alguna forma.

Los filtros presentan una gran variedad de efectos producidos sobre una imagen, desde cambios de intensidad de color, cambios de tonos de color, cambios de brillo y contraste, hasta otros más interesantes como la detección de bordes, difuminación, o suavizado entre muchos más.

Los filtros utilizados en el TFG han sido:

- Box Blur
- Neighboring Middle Point
- Bilateral
- Alpha-Trimmed Mean

A continuación, serán explicados en detalle, junto con alguno más.

Kernel

Antes de comenzar con los filtros es necesario definir un elemento a menudo empleado en el procesamiento de imágenes. Un “kernel”, “matriz de convolución” o “máscara”, es una matriz (comúnmente cuadrada, aunque no tiene por qué serlo, ya que las hay circulares) de números enteros y normalmente de dimensión 3x3 o 5x5. Su tamaño indica la dimensión del entorno de vecindad empleado en el filtrado de la imagen, es decir, en el caso de 3x3, significa que se utilizarán 9 píxeles para el procesamiento de un único pixel, en concreto el central.

La forma de la que se aplica es la siguiente: Todos los píxeles del entorno de vecindad se multiplican por su elemento correspondiente del kernel y a continuación se suman todos los valores obtenidos. Supongamos un píxel en la posición horizontal “x”, y vertical “y”, es decir (x, y). A continuación, vamos a aplicar el kernel de “Identidad” al pixel localizado en la posición (x, y) y veremos por qué este kernel nos da como resultado la misma imagen tras tratar todos sus píxeles.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Kernel de identidad[5]

El pixel localizado en (x-1, y-1) multiplicará al kernel en (1,1), el píxel (x, y-1) multiplicará al kernel en (1,2), y así sucesivamente con los 9 valores. Finalmente se suman todos los valores obtenidos por separado para obtener el nuevo valor que será asignado al pixel tratado.

Nos damos cuenta de que el kernel de identidad en concreto anula todos los valores de los píxeles vecinos al multiplicar por cero, dando como resultado del píxel central el mismo valor que poseía anteriormente (al multiplicar por uno), consecuentemente, obteniendo la misma imagen de salida que la de entrada.

Existe una gran diversidad de kernels, cada uno con un propósito, veremos alguno más posteriormente.

Problema o desventaja de los Kernels

Tras la introducción del kernel podemos preguntarnos: ¿Qué ocurre si estamos tratando un píxel que se encuentra exactamente en un borde de la imagen? Supongamos que estamos aplicando la matriz de convolución al pixel localizado en la esquina superior izquierda (píxel (0,0)), este píxel no tiene vecinos ni a su izquierda ni por encima suyo, por ello, no existe ningún valor por el cual multiplicar a la primera fila y columna de la matriz de convolución. Vamos a ver algunas soluciones.

Soluciones al problema del Kernel

1. Probablemente la solución más inmediata sería no tratar los píxeles de los bordes, comenzando a aplicar el Kernel en la primera posición que tenga todos los vecinos necesarios disponibles y dejando los pixeles no tratados a su mismo valor. Alternativamente, los pixeles no tratados se podrían poner al valor 0 generando bordes de color negro, o también se podrían eliminar de la imagen obteniendo una imagen de menor tamaño.

[5] https://en.wikipedia.org/wiki/Kernel_(image_processing)

2. Suponer que los píxeles que no tienen vecinos suficientes, tienen vecinos con valor 0. En el caso de un kernel de 3x3 y el pixel de la esquina superior izquierda, al cual le falta 5 vecinos, se le aplicaría el kernel suponiendo que los 5 vecinos que faltan tienen valor 0.

Como conclusión, independientemente de la solución elegida, siempre habrá un inconveniente, en la primera solución se pierden los bordes, y en la segunda se obtienen valores incorrectos para algunos píxeles al suponer que tienen vecinos con valor a 0, lo cual es falso. En el TFG se ha empleado la solución 1, dejando los bordes al mismo valor que tenían antes de aplicarles el filtro.

Filtro Box Blur y Gaussian Blur

Se trata de un filtro que pretende aproximarse al filtrado conocido como "Gaussian blur". Gaussian blur es el filtro más popular aplicado en procesamiento de señales digitales, por su alta capacidad de eliminar el ruido, o porque provoca un bonito efecto de "profundidad" en películas o videojuegos. No solamente es aplicado en imágenes sino también en otros tipos de señales, como por ejemplo las de audio. La gran desventaja del Gaussian blur es el coste computacional que presenta, ya que emplea operaciones exponenciales y la constante logarítmica "e", como vemos a continuación:

$$G(x) = \frac{e^{-\frac{x^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}$$

Gaussian blur en una dimensión[6]

$$G(x,y) = \frac{e^{-\frac{x^2+y^2}{2\sigma^2}}}{2\pi\sigma^2}$$

Gaussian blur en dos dimensiones. x y, indican las distancias al origen y σ es la desviación estándar.[7]

[6] https://en.wikipedia.org/wiki/Gaussian_blur

[7] https://en.wikipedia.org/wiki/Gaussian_blur

Podemos darnos cuenta de que en este caso concreto de Gaussian blur no se ha empleado un Kernel, sino una fórmula (función Gaussiana) que es aplicada a cada pixel en concreto, pero consta mencionar que sí existen Kernels que se aproximan al Gaussian blur:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Aproximación al Gaussian Blur con kernel 3x3[8]

Otro kernel que también se aproxima al Gaussian blur es el Box Blur, el cual resulta ser la media de los valores del entorno de vecindad:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Box Blur con kernel 3x3[9]

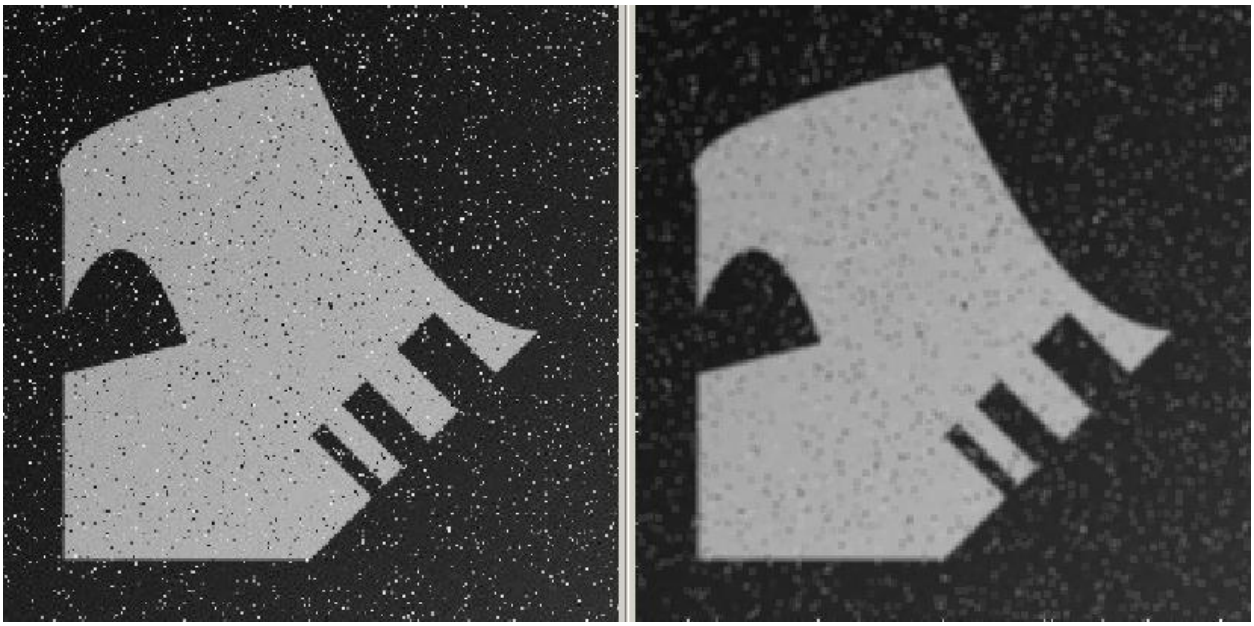


Imagen original con ruido sal y pimienta Imagen procesada con Box Blur 3x3

Podemos apreciar por la imagen anterior que el Box blur no elimina el ruido completamente, pero hace un trabajo bastante decente.

[8] https://en.wikipedia.org/wiki/Kernel_(image_processing)

[9] https://en.wikipedia.org/wiki/Kernel_(image_processing)

Neighboring Middle Point (Punto medio del entorno de vecindad)

Su base es hallar la semisuma de los píxeles de valor máximo y mínimo del entorno de vecindad. Este filtro disminuye mucho la nitidez, el cambio producido es bastante notable, y una de sus desventajas es que realza bastante los bordes a causa del pixelado que produce. El filtro es útil cuando se tiene una señal con poco ruido y se pretende reducir todavía más el ruido.

$$PMEV(x,y) = \frac{\max(i,j) + \min(i,j)}{2}, con\ (i,j)\ \epsilon\ S$$

Punto medio del entorno de vecindad, donde S es el entorno de vecindad.[10]

No es nada efectivo contra señales con alto ruido, pero realiza un trabajo bueno ante bajo ruido como se ve a continuación:

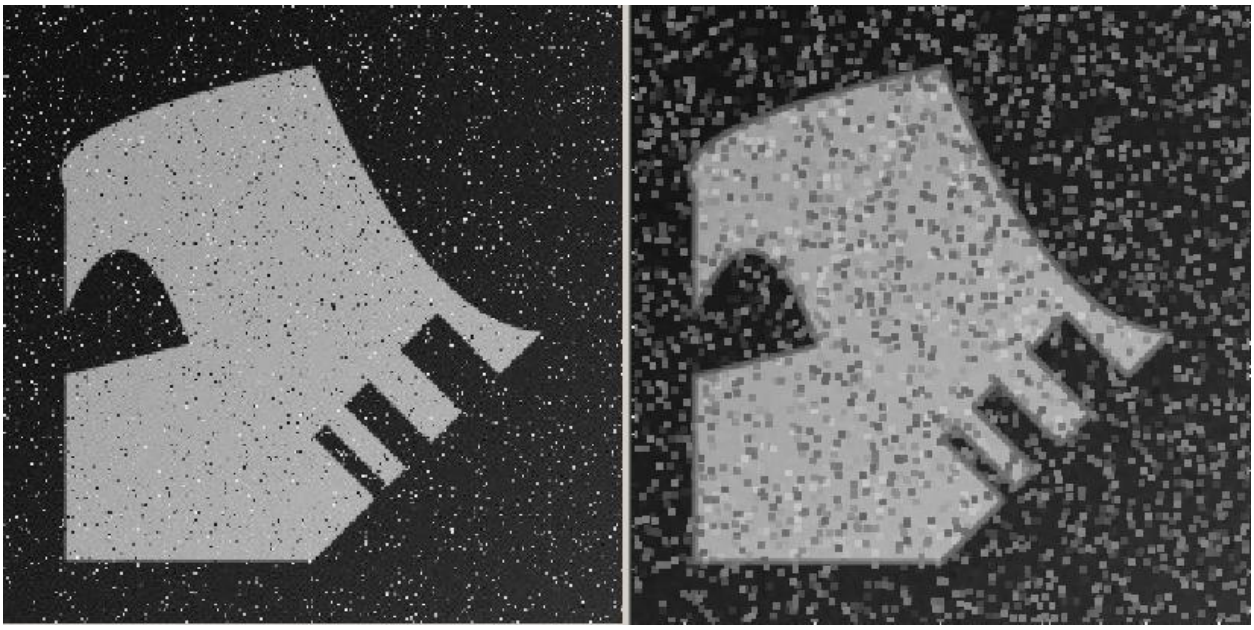


Imagen original con ruido sal y pimienta Imagen procesada con el filtro PMEV

En el caso de una imagen con mucho ruido incluso se podría decir que se ha obtenido un peor resultado comparado con la imagen original.

[10] http://titere.umh.es/tutorial/vision/cap5_2.htm

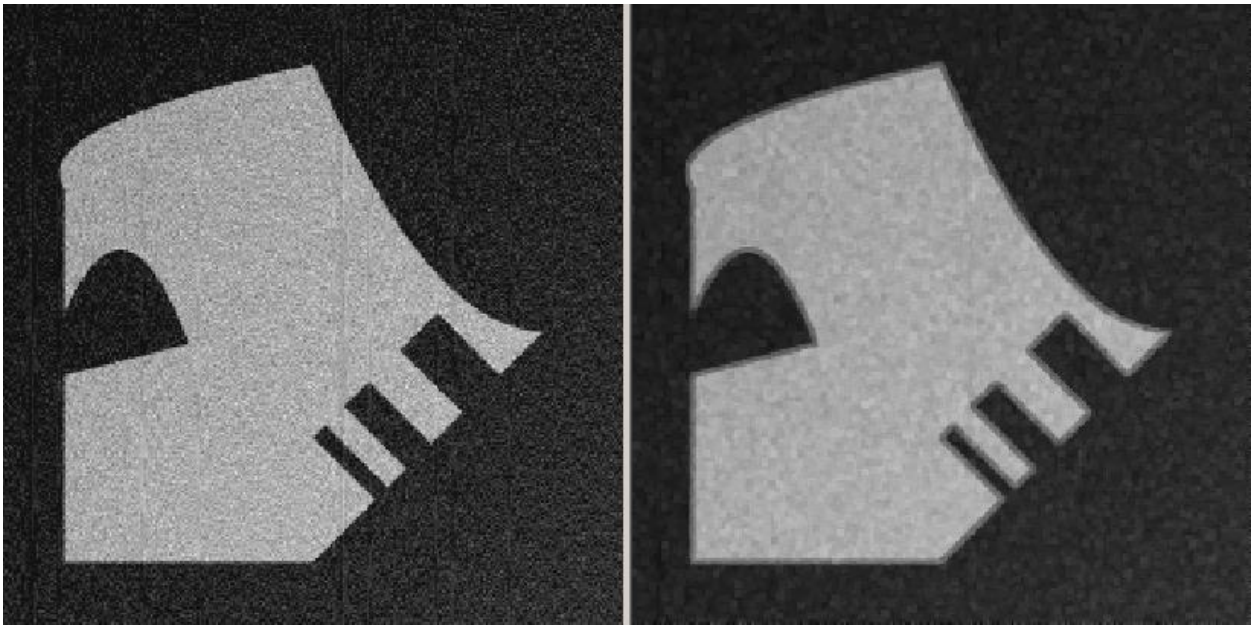


Imagen original con poco ruido sal y pimienta Imagen procesada con el filtro PMEV

En este segundo caso, puesto que la imagen posee poco ruido, en el resultado se ha obtenido una imagen más suave, en la cual el ruido es menos percibible.

Filtro Bilateral

Es un filtro bastante complejo el cual mantiene los bordes, y al mismo tiempo reduce el ruido y aplica suavizado. Tiene dos parámetros de entrada, sigma espacial "D" (cuanto mayor sea, más suavizada resultará la imagen), y sigma de rango "R" (cuanto mayor sea, más resultará en un filtro gaussiano). Para el TFG se han empleado los valores sigma D = 5 y sigma R = 50, los cuales aplican un "blureado" suave como se verá más adelante los ejemplos. Al igual que ocurría con el filtro Gaussian blur, el Bilateral también presenta un coste computacional bastante alto, (es por ello porque se le ha aplicado aceleración en el TFG). Consta de dos fases:

1. Se asigna un peso a cada pixel vecino, basado en la distribución Gaussiana anteriormente mencionada en el apartado "Filtro Box Blur y Gaussian Blur":

$$peso(i,j,k,l) = e^{\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}-\frac{\|I(i,j)-I(k,l)\|^2}{2\sigma_r^2}\right)}$$

Donde (i, j) y (k, l) es la distancia al origen del pixel tratado y todos sus vecinos respectivamente, e I (i, j) junto con I (k, l) es el valor o intensidad que toman los píxeles.[11]

2. Se realiza una normalización de los pesos para obtener el valor final:[12]

$$Intensidad(i,j) = \frac{\sum_{k,l} I(k,l) * peso(i,j,k,l)}{\sum_{k,l} peso(i,j,k,l)}$$

Veamos tres ejemplos del resultado producido por el filtro Bilateral:

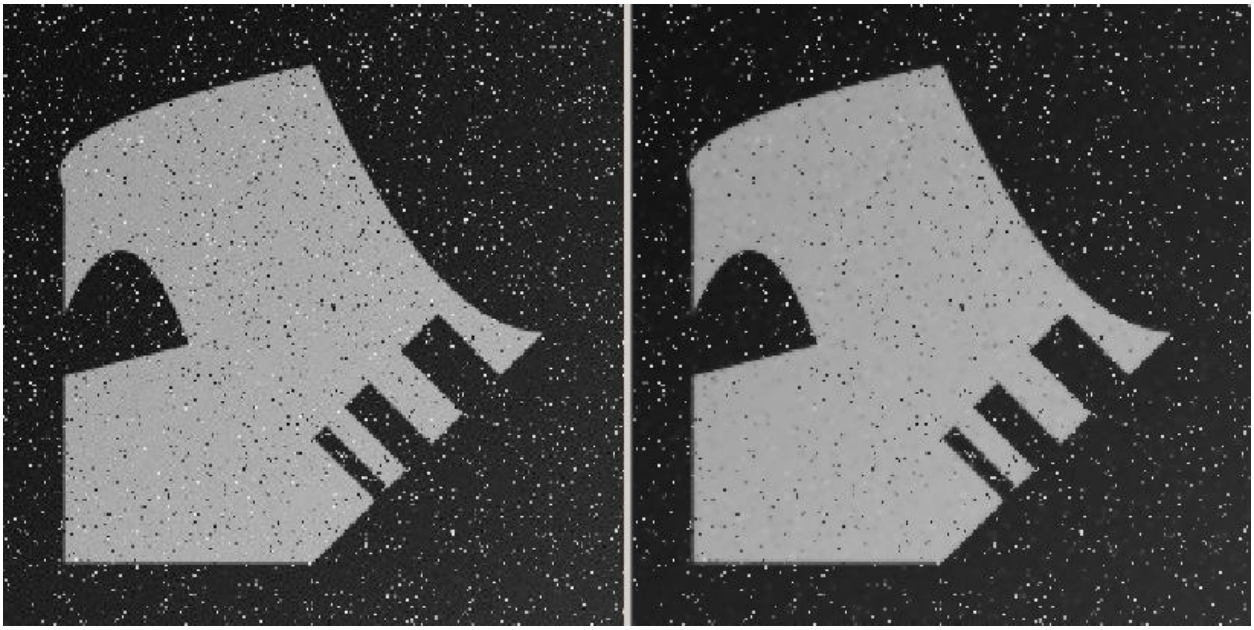


Imagen original con ruido sal y pimienta Imagen procesada con filtro Bilateral

En este ejemplo, el filtro no ha sido muy efectivo en las zonas oscuras (en las cuales el ruido se nota más), pero sí ha sido efectivo en la zona gris de la figura, como vemos se ha eliminado la mayoría del ruido blanco de la zona gris.

[11] https://en.wikipedia.org/wiki/Bilateral_filter

[12] https://en.wikipedia.org/wiki/Bilateral_filter

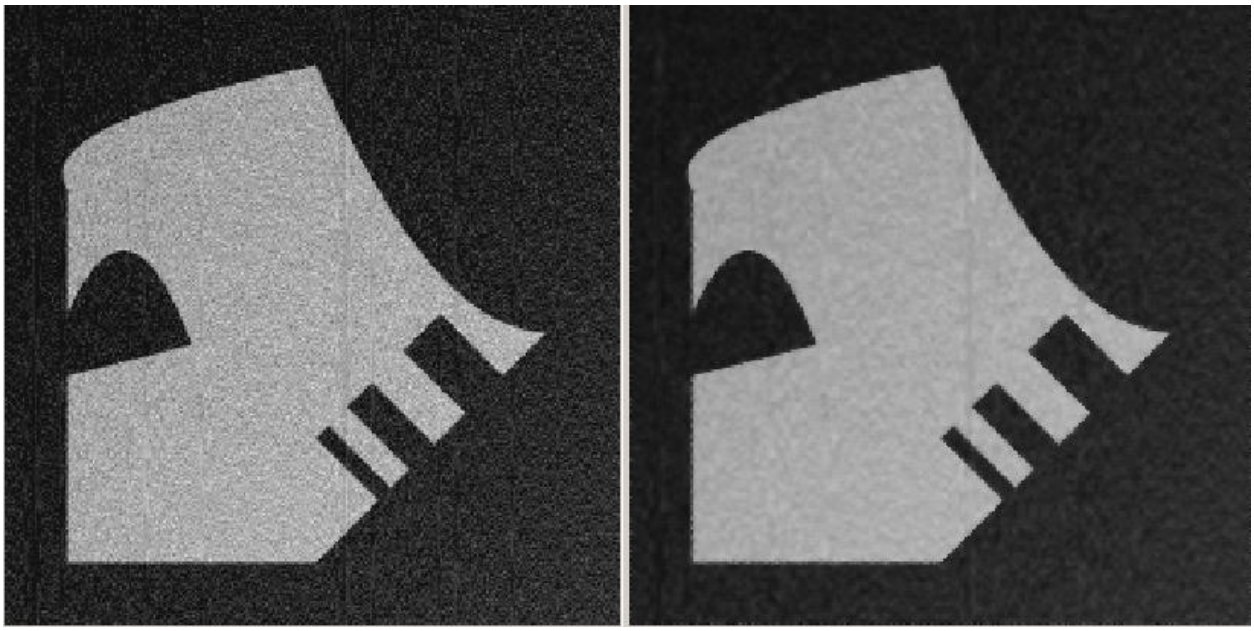


Imagen original con poco ruido sal y pimienta Imagen procesada con filtro Bilateral (1 iteración)

Como también ocurría con el filtro de “Punto medio del entorno de vecindad”, el filtro Bilateral ha realizado un mejor trabajo ante una imagen con poco ruido, y es incluso mejor que el obtenido en el Punto medio del entorno de vecindad.

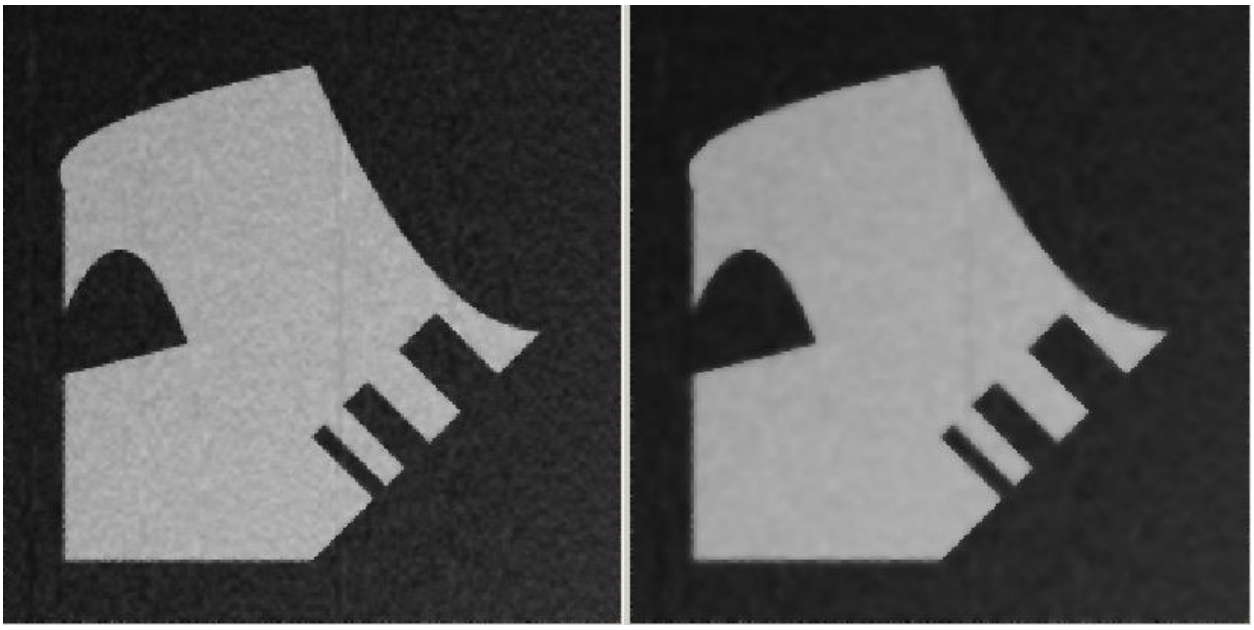


Imagen procesada con filtro Bilateral (1 iteración) vs. Imagen procesada con filtro Bilateral (4 iteraciones)

En este tercer ejemplo se demuestra que, tras varias pasadas, el filtro Bilateral es capaz de eliminar prácticamente todas las imperfecciones de una imagen y obtener un resultado muy suave.

Alpha-Trimmed Mean

Este filtro es una variación del "Box Blur" esto es porque realiza una media (como en Box Blur), con la diferencia de que la media es realizada eliminando P vecinos de menor y mayor valor. Esto supone realizar una ordenación de los píxeles vecinos para poder eliminar los P de menor y mayor valor. Puesto que la ordenación es muy costosa, el algoritmo ha sido acelerado en el TFG. Siendo N el tamaño del Kernel, el valor de P no puede ser mayor que (N*N) / 2. En el caso de un kernel de 3x3, P podría tomar los valores: [0, 1, 2, 3, 4]. Si P es 0 Alpha-Trimmed Mean coincidiría con el filtro Box Blur. Es de la forma:

$$Alpha - TrimmedMean(A) = \frac{\sum_{i=P}^{N*N-P} A_i}{N * N - 2P}$$

Donde A_i es la intensidad del pixel correspondiente del entorno de vecindad.[13]

La gran diferencia del Alpha-Trimmed Mean es que, con una única pasada y un valor correcto de P, consigue eliminar prácticamente el ruido por completitud. Veamos un ejemplo con P = 3:

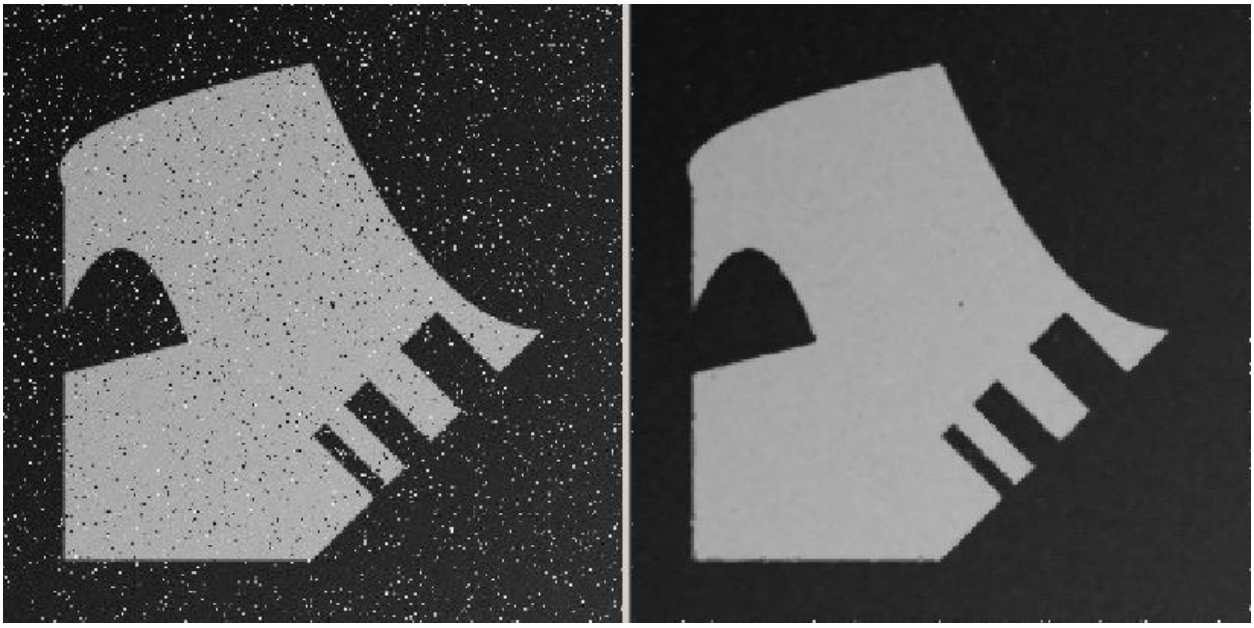


Imagen original con ruido sal y pimienta Imagen procesada con Alpha-Trimmed Mean P = 3

La efectividad de este filtro, como se puede observar, es bastante alta, puesto que ha conseguido mantener los bordes y eliminar el ruido.

[13] http://www.blackice.com/Help/Tools/Document%20Imaging%20SDK%20webhelp/WebHelp/Alpha-Trimmed_Mean_Filter.htm

Otros tipos de filtros

Como se mencionó anteriormente, existen filtros para una amplia variedad de operaciones sobre señales. Hasta ahora hemos visto cuarto filtros cuyo principal propósito es eliminar el ruido. Veamos otros tipos de filtros aplicados a imágenes.

Filtros para detección de bordes:

Estos filtros pretenden detectar puntos en la imagen que difieren suficientemente con respecto a sus vecinos como para formar un borde. Normalmente se suele indicar un umbral para indicar si un pixel es candidato a ser borde. Entre los filtros más conocidos de detección de bordes están el filtro de Sobel, Canny, Prewitt o Roberts Cross, los cuales emplean operaciones complejas para llevarse a cabo.

Pero también existen alternativas de Kernels que se aproximan a estos algoritmos, veamos algunos:

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

(Aprox. Kernel 1 Aprox. Kernel 2 Aprox. Kernel 3)[14]

Y ahora algunos ejemplos de filtros de detección de bordes para la siguiente imagen:

Imagen original[15]

[14] https://en.wikipedia.org/wiki/Kernel_(image_processing)

[15] https://quake.bethesda.net/en/champions/6jK5we9mIE02y8EMQs4KMY

Aprox. Kernel 1 Aprox. Kernel 2 Sobel

Filtros de acentuación

Estos filtros sirven para acentuar o realzar los detalles de la imagen aplicando un efecto de "afilado", "escalera" o "bordes de sierra", al contrario que los filtros Box Blur o Gaussian, los cuales "difuminan" o "suavizan".

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Un kernel de acentuación[16]

Necesidades de aceleración

La aplicación de un filtro a una imagen es un proceso muy repetitivo, como se ha explicado, se ha de recorrer todos los píxeles de la imagen y procesarlos uno a uno. Este proceso se realiza de forma secuencial por una CPU, es decir, no se procesa más de un único pixel a la vez.

Podemos darnos cuenta de que el procesado de cualquier píxel de una imagen, es independiente de todos los demás píxeles, por ello, no se necesita esperar a que se calcule el valor de un pixel en concreto para calcular otro pixel totalmente distinto.

[16] https://en.wikipedia.org/wiki/Kernel_(image_processing)

Lo que se acaba de mencionar es posible gracias a la computación paralela, cuyo principio está basado en la realización de operaciones masivamente paralelas e independientes entre sí. Existe un rango de dispositivos capaces de realizar estas operaciones, se trata de los dispositivos con capacidad de multi-threading, los cuales son capaces de procesar varios hilos de ejecución al mismo tiempo.

Entre otros, los más comunes son los dispositivos de procesamiento de gráficos (GPUs) y los procesadores con capacidad de multi-threading (CPUs con varios cores y threads por core).

Esta idea que se acaba de presentar parece muy interesante y es cierto que en muchos casos produce un altísimo impacto con respecto a tiempos de ejecución, pero no siempre es recomendable usar este tipo de tecnología en problemas paralelizables, ya que a veces provoca tiempos de ejecución paralela más lentos que los secuenciales. Esto es debido a que la programación paralela requiere una fase previa de inicialización y copias de memoria no necesarios en ejecuciones secuenciales.

Podemos concluir que se ha de hacer uso de la aceleración en los casos en los cuales el tiempo de inicialización + copias de memoria + ejecución, es menor que el tiempo de ejecución secuencial, para poder garantizar un beneficio.

Debido a que la mayoría de los filtros emplean operaciones sencillas de CPU, como la suma o multiplicación, este proceso funciona más rápidamente secuencialmente en CPU que en HW acelerado, esto es para dos de los filtros usados en el trabajo (Box Blur y Punto Medio del Entorno de vecindad).

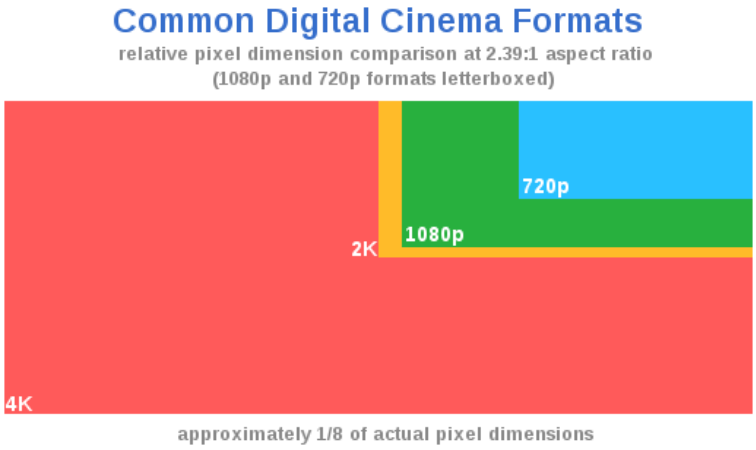
Por otro lado, el filtro Bilateral y Alpha-Trimmed Mean requieren de operaciones más complejas, como las exponenciales o la ordenación de elementos (que tienen coste lineal), por esta razón, estos dos filtros han sido candidatos a ser acelerados mediante el lenguaje de programación paralela OpenCL. En el Capítulo 4 veremos qué mejora con respecto al tiempo de ejecución secuencial ha supuesto la implementación paralela.

Capítulo 3 – Codificación de Video H.265/HEVC

En el mundo de la industria cinematográfica, a menudo es común que el metraje se consiga a resoluciones como UHD-2 (7680x4320) conocido coloquialmente como 8K, o UHD-1 (3840x2160) conocido como 4K, y además grabando a la máxima calidad de color 4:4:4 (Ver sección “Subsampling en YUV”). Esto se realiza para conseguir la mayor calidad posible de detalle en “raw”, es decir el metraje tal cual, manteniendo la completitud de fotogramas y sin ningún tipo de modificación para estas.

Estas resoluciones requieren de una altísima capacidad de procesamiento para poder ser visualizadas en tiempo real y mucho mayor ancho de banda necesario para la visualización en streaming, (i.e. a través de internet). Hoy en día no es nada común la visualización de vídeos a tales resoluciones, por ello, tras la obtención del metraje, se realiza un procesado posterior de los datos en raw para reducirlos a unas resoluciones aceptables como FHD (Full High Definition) 1920x1080, HD (High Definition) 1280x720, comunes en internet, o SD (Standard Definition) 720x480 para la televisión por cable.

Aparte de la cantidad de espacio que cada fotograma ocupa, entra en juego la frecuencia de reproducción, es decir, la cantidad de fotogramas grabados o visualizados por segundo. Consecuentemente cuanto mayor sea dicha frecuencia, mayor será el espacio físico ocupado en disco. Entre las frecuencias comunes en la actualidad se encuentran 24hz para las películas, 30hz para vídeos en internet y videojuegos, o 60hz y cada vez más frecuentemente 120hz para videojuegos.



Comparación entre las resoluciones cinematográficas más comunes.[17]

Estándares de codificación de vídeo

Como ya hemos visto en el Capítulo 2, una primera aproximación a la compresión de video es el "chroma subsampling" para la reducción/eliminación de color no realmente necesario, y como también se ha visto esto podría llegar a obtener incluso un 50% menos de información guardada por fotograma manteniendo prácticamente la misma calidad.

La compresión no para en la fase de subsampling porque existe otra fase posterior denominada "Codificación de vídeo". Se trata de software capaz de convertir contenido raw (no comprimido) a un formato comprimido, y viceversa, basándose en métodos aproximados de eliminación de información no necesaria o redundante.

Esta compresión normalmente se suele realizar de forma "lossy compression", lo cual quiere decir que, una vez comprimido el vídeo, éste carecerá de información presente en el vídeo original, el cual ya no será posible recuperar a partir del vídeo comprimido. Consecuentemente será necesario de menos espacio de almacenamiento para el vídeo comprimido y también de un menor ancho de banda para su reproducción, pero por otro lado se pierde calidad. Por esta razón los codificadores de vídeo nos permiten elegir el nivel de compresión que se desea aplicar para dar la posibilidad de obtener una proporción aceptable entre espacio/calidad, o ancho de banda/calidad.

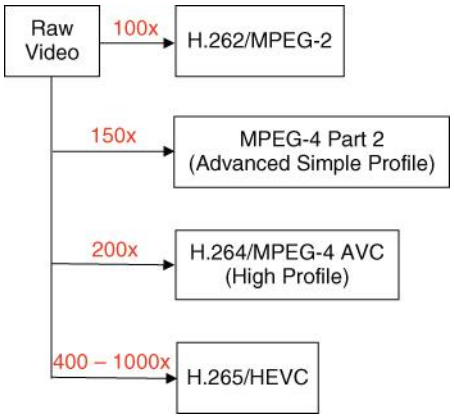
Por otro lado, el término "lossless compression", al contrario de "lossy compression" permiten la perfecta reconstrucción del archivo original a partir del comprimido. Pongamos como ejemplo el formato de archivo "ZIP", el cual realiza la compresión y descompresión exacta de los archivos. Como el propio término indica, no se pierde información y por esto, normalmente no se consigue ningún beneficio de espacio de almacenamiento, y en el caso de archivos de vídeo, una compresión lossless podría incluso incrementar el tamaño del archivo comprimido en contraste con el original.

[17] https://en.wikipedia.org/wiki/Digital_cinematography

La principal idea detrás de la compresión de video es encontrar información redundante en un fotograma (slice) o en una sucesión de fotogramas, y eliminar esta información guardándola solamente una vez para poder realizar la reconstrucción posterior.

Existe una amplia variedad de formatos de codificación de vídeo, el TFG está basado en el uso del estándar más actual, H.265/HEVC y en concreto de su uso mediante la implementación x265 en el programa FFMPEG.

H.256/HEVC



Capacidad de compresión del H.254/HEVC en comparación con sus predecesores.[18]

HEVC, o High Efficiency Video Coding es el estándar de compresión de video más actual, que ha conseguido doblar la eficiencia de su predecesor AVC (Advanced Video Coding), manteniendo la misma calidad que este último y obteniendo un 50% menor bit rate. Su primera versión completa fue desarrollada por MPEG y Video Coding Experts Group bajo el nombre de Joint Collaborative Team, aprobada por la ITU-T Recommendation, y publicada en 2013.

El principal motivo de su aparición ha sido la demanda de necesidad de una mayor eficiencia de codificación para las resoluciones más modernas de 8K y 4K, y también por el incremento de capacidad de procesamiento que los dispositivos multimedia presentan, para una mejor experiencia de entrega de contenido de alta calidad.

[18] Next-Generation Video Coding and Streaming by Benny Bing

H.264/AVC

AVC apareció por primera vez en 2003 como sucesor al H.262. Desarrollado por el Joint Video Team y estandarizado por la International Telecomunications Union, hoy en día es el estándar que domina la web debido a la gran eficiencia que presentó hasta la aparición del H.265 en 2013. Aunque sí es cierto que HEVC realiza un trabajo mucho superior, AVS se sigue utilizando debido a que todavía existe una amplia cantidad de dispositivos que no soportan la decodificación en HEVS, como por ejemplo las videocámaras personales, cámaras de seguridad, tablets o smartphones.

Predicción de fotogramas

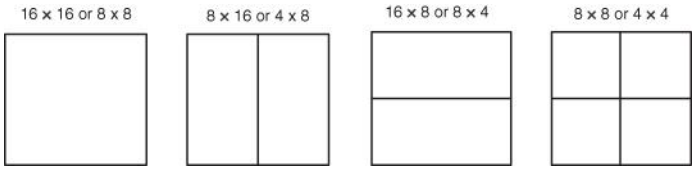
Tanto H.264 como H.265 emplean dos tipos de predicción de fotogramas para realizar la compresión, son las denominadas “Inter frame prediction” e “Intra frame prediction”.

Inter frame prediction

Se conoce como inter frame a un fotograma comprimido en base a la redundancia temporal de uno o varios fotogramas vecinos del mismo. Dado un fotograma, este se divide en bloques llamados macrobloques, y por cada uno de estos bloques, se intenta encontrar un bloque similar en un fotograma de referencia codificado anteriormente. Si la búsqueda tiene éxito, el bloque actual será codificado mediante un vector de moción, el cual es un vector bidimensional que apuntará a la posición del bloque similar en el fotograma de referencia, eliminándolo del fotograma actual.

Hay que tener en cuenta que en caso de que se encuentre un bloque buscado en un fotograma de referencia, no serán exactamente iguales, por ello se realiza un cómputo de diferencias residuales, las cuales serán utilizadas por el decodificador en el momento de la decodificación para realizar la reconstrucción del frame a partir del inter-frame codificado.

Si todo el proceso ocurre adecuadamente, el tamaño del vector de moción junto con las diferencias residuales de cada uno de los macrobloques, formará un inter frame de menor tamaño que el frame en raw, dando así lugar a la compresión.



Tamaño en píxeles de los macrobloques en H.264[19]

Intra frame prediction

Un intra frame es un fotograma el cual no necesita de información adicional para ser decodificado, al contrario de lo que ocurre con un inter frame. La predicción intra frame busca redundancia espacial entre los píxeles de un mismo fotograma, para realizar el cálculo de predicción de valores mediante la extrapolación de pixeles ya codificados mediante "delta coding". Delta coding es una forma de almacenamiento de datos en forma de *diferencias* (deltas) entre una secuencia de datos.

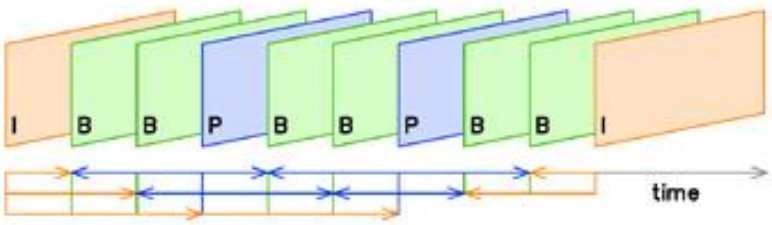
Tipos de fotogramas

Los dos modos de predicción de fotogramas anteriores forman tres tipos de fotogramas en función del modo de obtención del mismo:

- I frame: Son los fotogramas obtenidos mediante intra predicción. No necesitan información adicional para ser decodificados y son una fuente válida como frame de referencia para la inter predicción.
- P frame: Son los frames obtenidos unidireccionalmente de anteriores fotogramas P o I. Ocupan el 50% aprox. de lo que ocupa un fotograma I.
- B frame: Son los frames obtenidos bidireccionalmente de anteriores y posteriores fotogramas de referencia válidos. Ocupan el 25% aprox. de lo que ocupa un fotograma I.

Unos conjuntos de estos tres tipos de fotogramas en su forma comprimida forman lo que se denomina Group Of Pictures, el cual es de la forma IBBPBBP… Esta estructura presenta el problema de que es necesario calcular el cuarto fotograma antes del segundo y tercero, puesto que como ya se ha visto, para realizar la predicción de un fotograma B es necesario tener fotogramas válidos de referencia anteriores y posteriores.

[19] Next-Generation Video Coding and Streaming by Benny Bing



Ejemplo de un Group Of Pictures[20]

Métricas de calidad de vídeo

Debido a que para cada tipo de estándar de codificación existe multitud de implementaciones, aparece el concepto de medición de métricas de calidad para ver quién realiza un trabajo de codificación mejor. El concepto en el que se basan las métricas de calidad se puede entender con un ejemplo:

Supongamos que tenemos 300 fotogramas CIF (352x288) obtenidas en raw 4:2:0, los cuales forman un vídeo que ocupa aproximadamente 45 MB. Aunque no lo parezca, esta cantidad es bastante alta para la resolución CIF y la mínima cantidad de 300 fotogramas. Posteriormente realizamos dos codificaciones con HEVC, utilizando algunas de sus implementaciones, obteniendo dos vídeos de 250 KB y 300 KB.

Lo siguiente es decidir con qué codificador quedarnos. El tamaño que ocupan los vídeos no es una información suficiente para elegir un codificador ya que diferentes implementaciones producen distintos resultados, y no siempre el vídeo que ocupa más tiene mejor calidad, o el video que ocupa menos es mejor por el hecho de que ocupe menos espacio.

Aparte de la decisión subjetiva (realizando una comparación visual de los vídeos y decidir cuál nos gusta más) de qué codificador elegir, existen las métricas de calidad, las cuales informan acerca de diversos aspectos sobre un vídeo en raw y el vídeo producido por un decodificador tras realizar la codificación. Algunas métricas pueden interesar más que otras y no es necesario medirlas todas para llegar a una conclusión.

En el TFG se han usado y realizado la implementación de las siguientes métricas de calidad:

- PSNR

[20] https://en.wikipedia.org/wiki/Inter_frame

- SSIM
- DELTA
- MSE
- MSAD
- MSUBM
- MSUBFM

A continuación, se explicará el propósito y/o información aportada por cada uno de ellos.

PSNR

"Peak-to-peak signal-to-noise metric", es una de las métricas de calidad más comunes empleadas para medir la relación entre el ruido y la señal. Es útil para medir la calidad que se obtiene tras realizar la codificación, y cuanto mayor sea su valor, mayor será la calidad obtenida.

En este caso la señal es el vídeo original, y el ruido es el error producido por la diferencia de calidad entre el original y el codificado. Los valores comunes para vídeos con un "bit depth" de 8 bits se encuentran entre 30 y 50 decibelios, y para un bit depth de 16 bits, entre 60 y 80 decibelios.

$$PSNR = 20 * log_{10}(MAX_i) - 10 * log_{10}(\frac{\sum_{i=0,j=0}^{m-1,n-1}(I(i,j) - K(i,j))^2}{m * n})$$

Donde MAX_i es el máximo valor que puede tomar el bit depth (255 para un bit depth de 8); m y n son el ancho y alto de la imagen tratada respectivamente; I (i, j) y K (i, j) son los valores de los píxeles en el vídeo original y codificado respectivamente.[21]

En el caso de que se estén comparando dos vídeos idénticos, el valor PSNR será infinito porque la suma de las diferencias al cuadrado de los valores de los píxeles será 0, consecuentemente 0 / m*n = 0 y log10(0) = infinito.

SSIM

Otra de las métricas clásicas de medición de calidad que complementa a la PSNR. Útil para medir la similitud entre dos imágenes. Los valores que puede tomar están comprendidos entre 0.0 y 1.0, indicando completa desigualdad o completa igualdad respectivamente.

[21] https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

Para obtener el valor SSIM entre dos imágenes, éste se ha de calcular por partes, realizando ciertos cálculos sobre particiones de la imagen, es decir haciendo uso de una ventana de dimensiones NxN, normalmente 8x8.

Supongamos dos ventanas x e y sobre las que se quiere calcular el SSIM:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c1)(2\sigma_{xy} + c2)}{(\mu_x^2 + \mu_y^2 + c1)(\sigma_x + \sigma_y + c2)}$$

Fórmula SSIM[22]

Donde:

- $\mu_x = \frac{\sum_{i=0,j=0}^{N-1,N-1} I_x(i,j)}{N*N}$ = media de los valores de los píxeles en la ventana x.
- $\mu_y = \frac{\sum_{i=0,j=0}^{N-1,N-1} I_y(i,j)}{N*N}$ = media de los valores de los píxeles en la ventana y.
- $\sigma_x = \frac{\sum_{i=0,j=0}^{N-1,N-1} (I_x(i,j)-\mu_x)^2}{N*N}$ = varianza en la ventana x.
- $\sigma_y = \frac{\sum_{i=0,j=0}^{N-1,N-1} (I_y(i,j)-\mu_y)^2}{N*N}$ = varianza en la ventana y.
- $\sigma_{xy} = \frac{\sum_{i=0,j=0}^{N-1,N-1} \left((I_x(i,j)-\mu_x)*(I_y(i,j)-\mu_y)\right)}{N*N}$ = covarianza en las ventanas x e y.
- $c1 = (k_1 * L)^2, c2 = (k_2 * L)^2$ = variables necesarias para estabilizar el denominador.
- $L = 2^{\#bits\ por\ pixel} - 1$ = el máximo valor que un pixel puede tomar (255 para 8bits)
- $k_1 = 0.01, k_2 = 0.03$ por defecto.

Tras calcular todos los SSIM (x, y) que componen un fotograma se suman todos estos valores y se dividen entre la cantidad total de ventanas empleadas, para obtener el SSIM total entre el fotograma original y el codificado.

Estos cálculos satisfacen la simetría SSIM (x, y) = SSIM (y, x).

DELTA

Una métrica bastante más sencilla de implementar, al contrario que las anteriores, Delta no sirve para medir la calidad entre fotogramas, sino para comprobar dónde los codificadores

[22] https://en.wikipedia.org/wiki/Structural_similarity

provocan pérdidas de intensidades de luminosidad. Los valores de esta métrica están comprendidos entre -255 y 255, indicando el 0 que los dos fotogramas son idénticos.

$$DELTA(X,Y) = \frac{\sum_{i=0,j=0}^{m-1,n-1}(I_X(i,j) - I_Y(i,j))}{m * n}$$

Donde X e Y representan los fotogramas para los cuales se mide la métrica, m y n son las dimensiones de la imagen, e I (i, j) son las intensidades de los píxeles de los fotogramas.[23]

Un resultado positivo indica que el fotograma X posee más luminosidad que el fotograma Y, mientras que un resultado negativo indica lo contrario, y como ya se ha mencionado, el cero indicaría que no hubo cambios de intensidad entre los píxeles, por lo tanto, los fotogramas son idénticos.

MSAD

Muy parecida a la DELTA, empleada para probar la eficiencia de los codificadores o filtros. Los valores de esta métrica están comprendidos entre el 0 y 255, indicando el 0 que los fotogramas son idénticos.

$$MSAD(X,Y) = \frac{\sum_{i=0,j=0}^{m-1,n-1}|I_X(i,j) - I_Y(i,j)|}{m * n}$$

Donde X e Y representan los fotogramas para los cuales se mide la métrica, m y n son las dimensiones de la imagen, e I (i, j) son las intensidades de los píxeles de los fotogramas.[24]

Cuanto mayor sea el valor MSAD, más diferentes serán los dos fotogramas, con lo cual, los codificadores o filtros más eficientes darán como resultado un valor lo más cercano al cero por cada fotograma.

MSE

Esta métrica es empleada para evaluar la capacidad de un método o decodificador de reconstruir la imagen original a partir de la codificada. Los valores proporcionados por MSE están comprendidos entre 0 y 65025, indicando el 0 que los fotogramas comparados son idénticos.

$$MSE(X,Y) = \frac{\sum_{i=0,j=0}^{m-1,n-1}(I_X(i,j) - I_Y(i,j))^2}{m * n}$$

[23] http://compression.ru/video/quality_measure/info.html#delta

[24] http://compression.ru/video/quality_measure/info.html#msad

Donde X e Y representan los fotogramas para los cuales se mide la métrica, m y n son las dimensiones de la imagen, e I (i, j) son las intensidades de los píxeles de los fotogramas.[25]

Cuanto mayor sea el resultado proporcionado por MSE, peor es la reconstrucción de la imagen codificada, por ello, los codificadores más eficientes conseguirán obtener un MSE lo más cercano al 0 posible por cada fotograma decodificado.

MSU Blurring Metric

Muchos filtros y codificadores aplican blureado al realizar el procesamiento sobre los fotogramas, y esto a veces el espectador lo nota. Esta métrica permite medir la diferencia de intensidad de blureado entre dos imágenes. La diferencia con las anteriores métricas es que el cálculo se realiza para cada fotograma, sin depender de un fotograma de otro vídeo distinto.

Si el MSUBM obtenido en un fotograma del primer vídeo es mayor que el obtenido en el segundo, esto significa que el segundo fotograma está más blureado que el primero. Para realizar el cálculo del MSUBM se realiza (por cada fotograma) la media de una estimación de blureado calculado por cada píxel.

$$MSUBM(X) = \frac{\sum_{i=1,j=1}^{m-2,n-2}(|I_X(i-1,j) - I_X(i+1,j)| + |I_X(i,j-1) - I_X(i,j+1)|)}{(m-2)*(n-2)}$$

Donde X es el fotograma tratado; m y n son las dimensiones del fotograma; I (i, j) es la intensidad del píxel en la posición i, j del fotograma respectivo.[26]

MSU Brightness Flicking Metric

A diferencia de las métricas anteriores, esta únicamente se aplica al plano Y al ser el plano que contiene el brillo del vídeo, esto es porque, como el nombre de la métrica indica, mide los cambios de intensidad de brillo producido entre el fotograma actual y el anterior (fotogramas vecinos). Puesto que el primer fotograma (0) de cualquier vídeo no tiene ningún fotograma anterior, por defecto el MSUBFM (0) = 0.0. El MSUBFM de cada fotograma es independiente de fotogramas pertenecientes a distintos vídeos.

El cálculo se realiza haciendo la media de la suma de las diferencias absolutas de los píxeles entre los dos fotogramas (actual y a anterior).

---

[25] http://compression.ru/video/quality_measure/info.html#mse

[26] http://compression.ru/video/quality_measure/src/MSU_VQMT_Documentation.pdf

$$MSUBFM(X) = \frac{\sum_{i=0,j=0}^{m-1,n-1} |I_x(i,j) - I_{X-1}(i,j)|}{m * n}$$

Donde X es el fotograma tratado; m y n son las dimensiones del fotograma; I (i, j) es la intensidad del píxel en la posición i, j del fotograma respectivo.[27]

[27] http://compression.ru/video/quality_measure/src/MSU_VQMT_Documentation.pdf

Capítulo 4 – Resultados

En este capítulo se hará uso de la aplicación implementada en el TFG para realizar una serie de pruebas y valorar los resultados proporcionados por cada uno. Para ello se usarán dos vídeos distintos, uno con una cámara estática, en el cual no se produce mucha cantidad de movimiento, y un segundo con más movimiento de cámara y entorno.

Se realizarán 3 escenarios distintos por vídeo, empleando las características de configuración más comunes. Por supuesto, el usuario de la aplicación es libre de configurar los parámetros de ejecución, tanto para el vídeo a tratar, como para algunos filtros que requieren unos parámetros específicos.

Notas:

- Los datos de las métricas solo se muestran para el plano Y al ser el más importante, pero la aplicación calcula las métricas para todos los planos (con la excepción del MSUBFM como ya se ha explicado en el capítulo anterior por qué esto es así).
- Hay que tomar en consideración que los datos mostrados a continuación son en media por vídeo ya que no tendría mucho sentido mostrar los resultados de los 300 fotogramas por cada métrica (eso se puede ver en cualquier momento en los archivos de salida proporcionados por la aplicación).
- Los resultados de la métrica MSUBFM no son mostrados, porque realizar una media de los mismos no aportaría ninguna información relevante (si se quiere ver dicha información habría que ir al archivo en concreto de salida proporcionado por la aplicación implementada).
- Los vídeos usados son en formato YUV420 al ser el más común en el mundo digital, pero la aplicación implementada permite usar otros tipos de formatos como YUV422, YUV444, etc.
- Los codificadores x264 y x265 emplean lo que se conoce como QP (quantization parameter) para realizar la codificación. Este parámetro indica la cantidad de información que se ha de eliminar de un macrobloque, y una vez establecido el QP, la misma cantidad de información es eliminada de cada macrobloque. Por otro lado, el término CRF (Constant Rate Factor), permite codificar diferentes fotogramas a diferentes niveles de compresión. Por ejemplo, un CRF = 20 usará un QP = 20, pero

en las escenas con mucho movimiento podrá aumentar a QP = 22 para una mayor compresión, y en las escenas con poco movimiento bajará a QP = 18 para mantener una mejor calidad. En el trabajo se ha decidido dar al usuario la posibilidad de ajustar el CRF en vez del QP y será el término al que se hará referencia en las siguientes pruebas.

Vídeo 1 – hall_monitor_cif_420.yuv

Es un vídeo en formato YUV420 compuesto por 300 fotogramas.

Escenario 1

Ventana de los vecinos de 3x3 con un CRF de 20. Tamaño original del vídeo: **44550KB**.

Métrica/Filtro	Sin filtro	A-T Mean	Bilateral	Box Blur	PMEV(NMP)
Tamaño (KB)	388	281	272	**228**	253
SSIM	0.96	**0.93**	**0.93**	0.91	0.88
PSNR	38.77	30.86	**32.33**	28.71	26.37
DELTA	-0.05	**0.08**	0.42	0.53	0.74
MSAD	2.04	**3.58**	3.69	4.84	6.06
MSE	8.76	53.27	**37.96**	87.43	149.71
MSUBM in	20.45	20.45	20.45	20.45	20.45
MSUBM out	18.59	14.13	**14.61**	13.47	13.50

Como conclusión de este escenario, todas las métricas producen una altísima compresión con respecto al tamaño original (más del 99%), pero el Box Blur es el que más se ha comprimido. Si no se aplica ningún filtro, se obtiene un 96% de similitud entre los dos vídeos, pero de los filtros, los mejores con respecto al SSIM son el A-T Mean y el Bilateral con un 0.93. El filtro que menos ruido provoca es el Bilateral con un PSNR de 32.33. Puesto que todos los DELTA son cercanos al cero, las decodificaciones de los vídeos no provocan cambios de intensidad notables. El MSAD nos indica que la decodificación de los fotogramas en PMEV es la peor de todas al ser la mayor, con lo cual los fotogramas se parecerán menos que en los otros filtros. De nuevo, el MSE en PMEV es con mucha diferencia mayor a los demás, por ello la reconstrucción producida por el decodificador es muy pobre. Finalmente, puesto que todos los MSUBM out están por debajo del MSUBM in, se produce blureado en todas las codificaciones.

Escenario 2

Se vuelve a usar una ventana de vecinos de 3x3, incrementando la calidad del codificador a un CRF 18. Tamaño original del vídeo: **44550KB**.

Métrica/Filtro	Sin filtro	A-T Mean	Bilateral	Box Blur	PMEV(NMP)
Tamaño (KB)	637	430	410	**344**	389
SSIM	0.96	**0.93**	**0.93**	0.91	0.88
PSNR	39.60	30.98	**32.52**	28.78	26.38
DELTA	-0.03	**0.10**	0.44	0.56	0.75
MSAD	1.89	**3.48**	3.61	4.78	6.03
MSE	7.30	51.86	**36.37**	86.01	149.35
MSUBM in	20.45	20.45	20.45	20.45	20.45
MSUBM out	18.82	14.23	**14.70**	13.54	13.59

Con respecto a los resultados obtenidos anteriormente, incrementar la calidad del codificador a un CRF de 18 provoca la obtención de unos resultados de métricas muy parecidos a un CRF 20, con la excepción del incremento del tamaño del archivo. En este punto podemos ver que los candidatos a filtros más eficientes son el A-T Mean y el Bilateral, en términos de calidad/espacio.

Escenario 3

Como hemos visto en el ejemplo anterior, usar un CRF 18 frente a 20 no es nada rentable para un vídeo con cámara estática, por ello, en este escenario se volverá a usar CRF 20 pero empleando una ventana de vecinos de 5x5.

Métrica/Filtro	Sin filtro	A-T Mean	Bilateral	Box Blur	PMEV(NMP)
Tamaño (KB)	388	175	221	**168**	238
SSIM	0.96	0.83	**0.90**	0.82	0.72
PSNR	38.77	25.31	**30.12**	24.70	21.93
DELTA	-0.05	**0.24**	0.39	0.55	1.56
MSAD	2.04	6.62	**4.62**	7.28	10.17
MSE	8.76	191.04	**63.25**	220.14	416.31
MSUBM in	20.45	20.45	20.45	20.45	20.45
MSUBM out	18.59	10.74	**12.61**	10.63	11.02

Debido a la baja resolución del video y a la alta cantidad de píxeles vecinos empleados, se produce una pérdida significativa de calidad, y un trabajo pobre de reconstrucción por parte de los decodificadores, como podemos ver, el MSE de los filtros frente a no aplicar ningún filtro, son extremadamente altos. Un dato curioso es el SSIM del filtro Bilateral, ya que ha conseguido mantener el 90% de calidad y una pérdida de 6% frente a no aplicar ningún filtro, y también es el filtro que más PSNR ha conseguido. En este punto parece que el filtro Bilateral es el candidato en cabeza a ser el filtro más eficiente, y el PMEV el último, en términos de calidad/espacio.

Vídeo 2 - coastguard_cif_420.yuv

Es un vídeo en formato YUV420 compuesto por 300 fotogramas. A diferencia del primer vídeo, éste contiene movimiento de cámara.

Escenario 1

Ventana de los vecinos de 3x3 con un CRF de 20. Tamaño original del vídeo: 44550 KB.

Métrica/Filtro	Sin filtro	A-T Mean	Bilateral	Box Blur	PMEV(NMP)
Tamaño (KB)	1161	771	655	**521**	665
SSIM	0.96	**0.84**	**0.84**	0.81	0.77
PSNR	36.13	28.71	**29.58**	27.42	25.88
DELTA	0.00	0.36	0.50	0.45	**0.22**
MSAD	2.96	**6.12**	6.18	7.38	8.70
MSE	16.33	88.40	**72.27**	119.19	169.92
MSUBM in	31.27	31.27	31.27	31.27	31.27
MSUBM out	29.30	18.94	**19.08**	17.22	16.71

Debido a que se trata con un vídeo más complejo, incluso los dos mejores filtros no consiguen obtener más de SSIM 0.84 (es decir se produce un 16% de pérdida de calidad en el mejor caso). Si tomamos en consideración el A-T Mean y el Bilateral, los dos consiguen métricas muy parecidas con la excepción del tamaño final del archivo, para lo cual el Bilateral gana.

Escenario 2

Volvamos a incrementar la calidad de codificación a un CRF 18 para ver si se puede obtener una mejora de calidad en este vídeo. Ventana de los vecinos sigue siendo de 3x3.

Métrica/Filtro	Sin filtro	A-T Mean	Bilateral	Box Blur	PMEV(NMP)
Tamaño (KB)	1532	1028	856	**676**	911
SSIM	0.97	**0.85**	**0.85**	0.82	0.77
PSNR	37.50	28.89	**29.83**	27.55	25.93
DELTA	0.00	0.36	0.50	0.46	**0.22**
MSAD	2.55	**5.91**	6.00	7.24	8.63
MSE	12.00	84.89	**68.20**	115.64	168.31
MSUBM in	31.27	31.27	31.27	31.27	31.27
MSUBM out	29.75	19.18	**19.29**	17.40	16.92

El SSIM incrementa únicamente en un 1% para A-T Mean y Bilateral, lo cual no beneficia mucho frente al incremento de tamaño que también se produce. Un aspecto interesante es que los resultados del PMEV prácticamente son los mismos que con CRF 20, pero con un mayor tamaño de archivo, por ello esta métrica demuestra ser la menos eficientes de todas en todos los aspectos (ver que ocupa más que otras métricas y es la que menos calidad aporta).

Escenario 3

Como hemos visto, un incremento de calidad de CRF 18 no es nada rentable, veamos cómo afecta usar una ventana de vecinos de 5x5 con CRF 20.

Métrica/Filtro	Sin filtro	A-T Mean	Bilateral	Box Blur	PMEV(NMP)
Tamaño (KB)	1161	375	481	**365**	617
SSIM	0.96	0.61	**0.71**	0.60	0.51
PSNR	36.13	23.77	**27.03**	23.43	21.75
DELTA	0.00	0.54	0.52	0.49	**0.01**
MSAD	2.96	10.69	**8.30**	11.20	13.58
MSE	16.33	276.97	**129.90**	299.60	441.02
MSUBM in	31.27	31.27	31.27	31.27	31.27
MSUBM out	29.30	11.52	**14.03**	11.33	11.99

En definitiva, estos han sido los peores resultados obtenidos en los escenarios, por ello podemos concluir que una ventana de vecinos de 5x5 no es nada eficiente frente a vídeos donde se producen muchos movimientos. Independientemente de lo dicho, el filtro Bilateral es que mejor calidad-espacio-ruido ha conseguido en este escenario.

Aceleración Hardware

Para concluir el capítulo, veamos como son afectados los tiempos de ejecución de los escenarios 3, de ambos vídeos. Elegimos el escenario 3 porque es el que emplea una dimensión de entorno de vecindad de 5x5, por ello los tiempos de ejecución serán un poco más notables que con un entorno de 3x3.

- Vídeo 1: Tiempo secuencial: 36s; Tiempo en GPU: 33s; Por la Ley de Amdahl: (Aceleración = Tiempo secuencial / Tiempo paralelo), podemos calcular Aceleración = 36 / 33 = 1.09 = 1 + (9 / 100), de donde se concluye que se ha obtenido una aceleración únicamente del 9% frente al cómputo paralelo. Esto no es una cantidad muy buena pero como se mencionó en el Capítulo 2, la aceleración hardware no siempre produce los mejores resultados, y además hay que tener en cuenta que se está tratando con una dimensión de fotogramas de 352x288, la cual

es muy pequeña para obtener un notable beneficio con respecto a la ejecución secuencial.

- Vídeo 2: Tiempo secuencial: 40s; Tiempo en GPU: 35s; Aceleración = 40 / 35 = 1.14 = 1 + (14 / 100), obteniendo una aceleración del 14% frente al cómputo en paralelo. Debido a que se trata de un vídeo con más movimiento, y más complicado de procesar, se ha obtenido un mayor beneficio frente al vídeo anterior.

Conclusión de las pruebas

Con respecto a los resultados obtenidos de las pruebas, podemos realizar una ordenación de los filtros por la eficiencia que presentan de calidad/espacio, siendo el primero el más eficiente:

1. Bilateral
2. Alpha-Trimmed Mean
3. Box Blur
4. Neighboring Middle Point

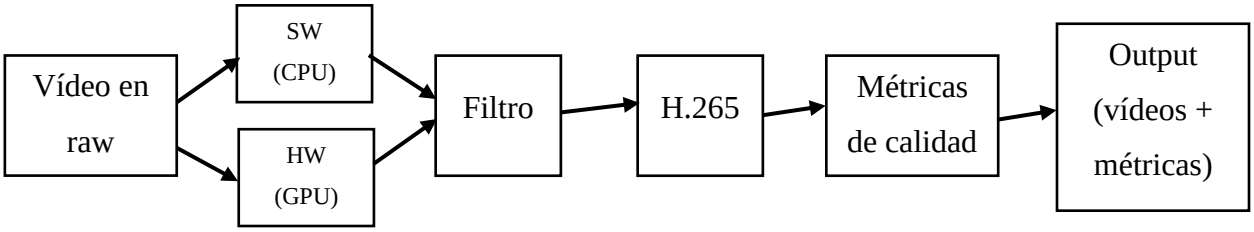
Consta mencionar que, si se pretende obtener la mayor calidad independientemente del tamaño final obtenido, la mejor opción es no aplicar ningún filtro, puesto que de esta forma se obtienen los mejores resultados en las métricas y la mejor calidad de codificación.

Capítulo 5 – Conclusión y Trabajo Futuro

Conclusión

Como acabamos de apreciar en las pruebas, los codificadores de vídeo resultan ser elementos cruciales para la reproducción de contenido multimedia a través de internet, gracias a la capacidad que presentan de comprimir altas cantidades de bit rate (y tamaño), a cambio de un compromiso casi despreciable de calidad de vídeo.

Para concluir, se puede realizar un esquema para tener una visión global del trabajo realizado:



La aplicación implementada se ha dejado como un módulo a partir del cual el usuario puede determinar qué filtro seleccionar en función de ciertos parámetros o necesidades.

Trabajo Futuro

Se nos presenta varios aspectos en los cuales el trabajo podría extenderse o mejorarse:

1. Considerando los detalles técnicos de la implementación de la aplicación desarrollada para el TFG, se podría mejorar gran cantidad de aspectos, como por ejemplo más comprobación de errores, la incorporación del programa "ffmpeg" dentro de la aplicación, sin necesidad de hacer una llamada externa al mismo para la codificación, o incluso plantearse la realización de interfaz gráfica más simple que la de "MSU Video Quality Measurement Tool", entre otros.
2. Implementar una parte de la aplicación que no simplemente calcule las métricas correspondientes entre los vídeos, sino que también aporte feedback subjetivo al usuario de cuáles han sido los mejores resultados obtenidos (a lo mejor en función de unos parámetros de entrada proporcionados por el usuario indicando qué es lo que se está buscando: calidad, espacio, eficiencia de codificador, etc.) sin necesidad

de que el usuario tenga que consultar todos los archivos de salida proporcionados por la aplicación, y también, posiblemente ordenar los resultados por métricas de los distintos filtros.

3. En un principio, para el presente TFG se decidió realizar experimentos con métricas obtenidas de usuarios reales en función de los distintos filtros, no simplemente aportadas por la aplicación. Estas métricas son conocidas como métricas subjetivas, pero debido a la falta de tiempo y más complicaciones que esto supondría, se decidió no realizar estos experimentos, con lo cual esto sería otra posible extensión del trabajo.
4. MSU Video Quality Measurement Tool permite la medición de más filtros aparte de los implementados en este trabajo, como el "MSU Drop Frame Metric" el cual calcula si el decodificador ha producido alguna pérdida de frames con respecto al vídeo original no codificado, o por ejemplo la métrica "MSU Scene Change Detector" que detecta en qué momento se produce un cambio totalmente distinto de escenario en el vídeo. Estas dos métricas que acabamos de mencionar no habrían aportado nada de información desde el punto de vista espacio/calidad (que es de lo que el TFG trata), por ello no han sido implementadas, con lo cual, otra posible extensión de la aplicación sería implementar más diversidad de métricas que no sean relacionadas únicamente con la calidad producida por los codificadores.

Referencias y Bibliografía

Capítulo 1

- http://www.internetworldstats.com/emarketing.htm
- http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

Capítulo 2

- https://en.wikipedia.org/wiki/Chroma_subsampling
- https://en.wikipedia.org/wiki/RGB_color_model
- https://docs.gimp.org/en/plug-in-convmatrix.html
- http://titere.umh.es/tutorial/vision/cap5_2.htm
- http://perso.telecom-paristech.fr/~ytendero/documents_aic/bilateral_filter.pdf
- https://en.wikipedia.org/wiki/Euclidean_distance
- https://people.csail.mit.edu/sparis/bf_course/course_notes.pdf
- http://www.blackice.com/Help/Tools/Document%20Imaging%20SDK%20webhelp/WebHelp/Alpha-Trimmed_Mean_Filter.htm
- OpenCL Programming by Example by Ravishekhar Banger; Koushik Bhattacharyya
- Heterogeneous Computing with OpenCL, 2nd Edition by Dana Schaa; Perhaad Mistry; David R. Kaeli; Lee Howes; Benedict Gaster

Capítulo 3

- http://slhck.info/video/2017/02/24/crf-guide.html
- Next-Generation Video Coding and Streaming by Benny Bing (2015)
- https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding
- https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC
- https://en.wikipedia.org/wiki/Video_coding_format
- https://en.wikipedia.org/wiki/Video_codec
- https://en.wikipedia.org/wiki/Inter_frame

- https://en.wikipedia.org/wiki/Intra-frame_coding
- http://compression.ru/video/quality_measure/info.html#start